Multisolution of Complex Performability Models in the OsMoSys/DrawNET Framework

Gribaudo Marco*, Mazzocca Nicola†, Moscato Francesco‡, Vittorini Valeria†
*Dipartimento di Informatica,
Universita' di Torino, corso Svizzera 185, Torino, Italy,
Email:marcog@di.unito.it
†Dipartimento di Informatica e Sistemistica,
Universita' degli Studi di Napoli "Federico II", Via Claudio 21, 80125 Napoli, Italy,
Email: {n.mazzocca,vittorin}@unina.it
‡Dipartimento di Ingegneria dell'Informazione,
Seconda Universita' di Napoli, Via Roma 29, 81031 Aversa, Italy,
Email: francesco.moscato@unina2.it

***Abstract*— In the last years some infrastructures and frameworks have been proposed to enable the compositional development of multiformalism models. This effort requires proper techniques and tools to guarantee generality and flexibility when combining multiple solvers and results obtained from the analysis of such models. The OsMoSys/DrawNET framework allows to develop and analyze complex performability models which are composed by several submodels expressed by means of different formal languages. In this paper we describe the approach to multisolution of multiformalism models in the OsMoSys/DrawNET framework and we introduce the mechanisms used to define the performance indices and generate the required results.**

A raid system is used to exemplify the solution process of a performability model from its creation to the analysis and the presentation of the results to the final user.

I. Introduction

Modern industrial systems are built up of heterogeneous components which interact to perform the required tasks under real time and dependability or safety constraints. The increasing complexity of such systems requires the availability of proper methods and techniques for their specification and analysis. To cope with complexity, multiformalism approaches are emerging that allow to combine multiple modeling methods into an unified framework, in order to enable the compositional development of models without imposing a-priori what types of formalisms would be used by the designer (e.g. see [6], [7], [21]). In multiformalism modeling each system component to be represented may be expressed through the most suitable formalism, and proper infrastructures should be available to support multiple interacting formalisms and solvers. One of the challenge when exploiting multiformalism is to guarantee the generality and the flexibility required to solve the resulting models and to combine multiple solvers and results. In this paper we address the problems that arise when handling the indices defined by the user on a multiformalism model if a very general approach to multisolution is adopted that does not bring back the various formalisms to a common semantics or interface. We introduce an original approach to multisolution that has been implemented in the OsMoSys/DrawNET framework. OsMoSys is the name of both a modeling methodology and a software framework aimed at the integration and interoperability of different modeling and analysis techniques and tools [21], [12]. DrawNET is an autoconfiguring GUI for the development of multiformalism models at the user's level [11]. The basis for the tool configurability is a metaformalism used to define the elements of any formalism a user may want to use. DrawNET allows to implement all the features of the OsMoSys modeling methodology and provides both a graphical representation and an XML (eXtended Markup Language) based description of the models. In the following we refer to the framework resulting from the integration of OsMoSys and DrawNET.

The paper is organized as follows. Section II provides a discussion about the related work. A Redundant Array of Inexpensive Disks (RAID) system is proposed as a running example in Section III. Section IV presents the OsMoSys/DrawNET approach to multisolution: the concept of orchestration is introduced and its application to the interaction of both modeling techniques and analysis/simulation tools is described. The mechanisms used to handle performance indices and results in the multisolution environment are defined in Section V. Finally Section VI contains some closing remarks.

II. Related Work

A few years ago, multisolution tools were required to provide several solution methods for solving models expressed through a specification language. The research progress in the field of system modeling and analysis led to the evolution of multisolution approaches due first to the compositional and hierarchical nature of complex models and then to the multiformalism modeling approaches developed to cope with the increasing complexity of systems.

The emerging research field of Computer Automated Multiparadigm Modeling (CAMPaM) [13] is trying to combine together the notions of *multiformalism modeling* and *metamodeling*. These modeling methodologies allow to face some of the modeling problems of complex, heterogeneous sytems

by coupling each modeling aspect with the best suitable formalism. In addition, a way to achieve flexibility for a modeling language to support many formalisms is to model the language itself with some meta-modeling methodology. Since these methodologies are emerging, it is necessary to face with all problems related to multiformalism modeling, in respect of the solution phase of this kind of models, by allowing solution tools to interact in order to automatically solve complex multiformalism models.

Consequently, a more general notion of multisolution is emerged that is the possibility of using more interacting solution techniques and tools to analyze models consisting of heterogeneous submodels, i.e. models obtained by composing parts that may be expressed through multiple interacting formalisms.

Efforts made to integrate multiple formalisms and solution methods within an unified framework have produced some important results. SHARPE and SMART are software tools that can be considered a first step towards the development of multiformalism multisolution frameworks.

SMART [4], [5] integrates Stochastic Petri Nets, Discrete-Time and Continuous-Time Markov Chains in a single modeling study for reliability and timing analysis. Submodels are solved using different solution techniques, including numerical methods and simulation, and they may exchange results through fixed point iterations. SHARPE [16], [18] is a tool for reliability and performability analysis. It allows to combine different model types (e.g. Fault Tree, Generalized Stochastic Petri Nets, Product Form Queuing Network, Markov Generative Process, etc.) and it provides flexible mechanisms for combining results so that models can also be developed by using hierarchical composition.

Möbius [6], [17] takes a more general approach than SMART and SHARPE, providing an extensible infrastructure to support multiple interacting modeling formalisms and solvers without presupposing what formalisms would be considered and what methods would be used to combine submodels. Nevertheless, formalisms have to be compatible with the framework since the user's models are translated into equivalent models using Möbius framework components. Models and solution techniques interact with one another through an Abstract Functional Interface, allowing them to interact with the framework components. Möbius is actually oriented to the application of solution methods based on the generation of (all or some of) the possible evolutions of the model in its state space.

A completely different solution is implemented by the DEDS and CADP toolboxes. They are "toolbox" by the sense of combining various tools and interfaces.

The DEDS toolbox [2] is oriented to the construction of modular tools for functional and performance analysis of Discrete Event Dynamic Systems. The main components of the toolbox are a GUI and various analysis programs that cooperate via a textual interface. The GUI allows to use and combine different modeling formalisms (Queuing Networks, Generalized Stochastic Petri Nets, Coloured Petri Nets) and it transforms the user's models into a (common) abstract Petri Net notation. CADP [8] is a toolbox for protocol engineering. It is more domain oriented and it allows to write protocol specifications written by means of different languages (Lotos, Labeled Transition Systems, SDL, μCRL, UML Real Time) and provides several tools for simulation, verification by equivalence/preorder checking, temporal logic model checking and test generation.

Another example of toolbox is the MODEST modeling tool for reliability and performance analysis of embedded systems [3]. It implements a singleformalism multisolution approach since MODEST is a process algebra used as an overarching notation for different model types (ordinary finite-state automata, timed automata, discrete event stochastic processes, Markov Decision Processes, etc.). The MODEST tool allows different external tools to cooperate. The external tools are integrated into the environment by developing proper adaptor modules called *satellite modules*. In [9] the authors say to have implemented a satellite module to translate MODEST specifications into the Abstract Functional Interface of Möbius in order to incorporate MODEST as a new atomic model specification formalism.

In this paper we propose an approach to multisolution based on the concept of orchestration and on the development of a set of XML-based languages. The proposed approach has been integrated in OsMoSys and in the DrawNET tool. OsMoSys consists of a modeling methodology to build multiformalism models and an open architecture for the integration of solution techniques and tools. The architecture of OsMoSys has some points of contact with MODEST: tools are integrated into the framework by means of proper wrappers called *Adapters* and a core module is in charge of executing the steps needed to solve and analyze the models. Nevertheless the OsMoSys approach to multisolution is more general: it allows very different solution techniques and tools to cooperate if the semantics of the their integration has been defined. The core module is a workflow engine [12] that takes as inputs a model description and the user definition of the performability indices generated by DrawNET, and executes the steps needed to produce the results according to an algorithm that implements the integration semantics. This requires that proper mechanisms and tools are defined and implemented to enable the communication between the user level in which the indices are expressed and the OsMoSys level in which the results must be evaluated.

III. A RAID Case Study

This Section introduces a running example that will be used throughout the paper to describe the approach to multisolution in the OsMoSys/DrawNET framework. At this aim we have chosen a simplified Raid level 5 system. Raid Systems [14] are high performance secondary storage systems which are designed to be fault-tolerant by storing redundant data on extra disks. The redundancy can be achieved by providing an identical copy of each disk (*mirroring*) or by means of a *parity disk*.

The simplified architecture we have adopted in this paper uses parity and can tolerate one disk failure [10]. The system may be in three different states:
a) *Ok* (the system is working), b) *Degraded* (the system is working despite of one disk failure) and c) *Dead* (the system is not working since two or three disk failures have occurred). Two controllers are the core of the system: the disk array controller which enqueues the service requests (read and write operations) and redirects them to the disk array, and the controller located on the physical disk array machine.

We want to build a performability model of this system that allows to evaluate the mean response time of read and write operations (RWMRT) while the system is in the state *Ok* or *Degraded*. Thus qualitative and quantitative aspects of the system must be considered. A multiformalism approach to the modeling of this system allows to cope with this requirement and also promotes a compositional approach by using reusable submodels. In this case the RWMRT can be evaluated by solving a model of the system consisting of three queues, expressed by means of the Queuing Network (QN) formalism and augmented by information obtained by solving two Fault-Trees (FT) submodels and two Generalized Stochastic Petri Nets (GSPN) submodels. FTs are used to evaluate the probabilities of the *Degraded* and *Dead* states, GSPNs are used to evaluate the service time of the disk array controller.

In the following FT_{DEAD} and FT_{DEGR} denote the FT models used to evaluate the reliability of the disk array in *Dead* or *Degraded* state respectively. They require that the mean time between failures (MTBF) of the disks is known. FT_{DEAD} is shown in Fig.1. The occurrence of the failure represented by

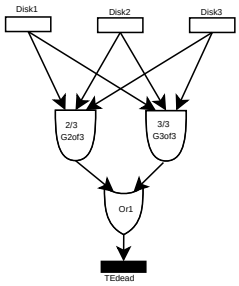

Fig. 1. FT_{DEAD} Fault Tree model of the RAID system

the *top event TEdead* depends on the probability that at least two over three disks fail. The *basic events Disk1*, *Disk2* and *Disk3* are parametrized by using the MTBF of the disks. For brevity's sake we omit to describe FT_{DEGR}.

Analogously, $GSPN_{OK}$ and $GSPN_{DEGR}$ denote the GSPN models used to evaluate the service time of the controller (in *Ok* or *Degraded* state respectively) that in turn depends on the algorithms used in read and write operations.

$GSPN_{DEGR}$ is shown in Fig.2. It describes the behavior of the raid system while it is working in a *Degraded* state (i.e. with one disk failure). Write (read) operations are modelled on the top and middle part (bottom) of the figure. Data are striped

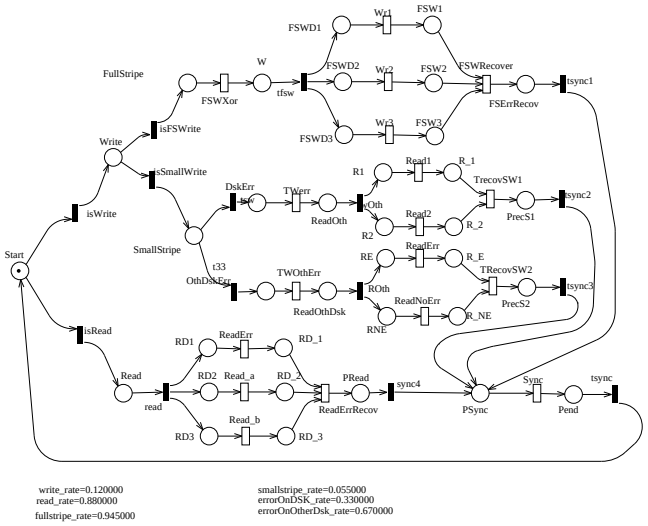

Fig. 2. $GSPN_{DEGR}$ model of the RAID system

along the three disks of the array during a write operation [14]. Write operations may write a full stripe (top of Fig.2) or they may be small stripe operations that only write a part of the full stripe (middle of Fig.2).

During a full stripe write, the parity data are evaluated (transition $FSWXor$) and the whole stripe (with parity) is written on the array. Since the array is in *Degraded* state, one of the write operation fails and the full stripe write ends.

To perform a small stripe write it is necessary to distinguish if the damaged disk is the disk on which the small stripe has to be written (transition $DskErr$) or the other disks (transition $OthDskErr$). Then read operations on the other disks are performed in order to evaluate the parity and to write the new stripe on the array. Full stripe writes have an higher rate then small stripe writes due to the caching mechanism embedded on the array. For the same reasons, only full stripe reads are performed on the array ($read$) and, in the *Degraded* state a recover operation is ever needed to accomplish read tasks ($ReadErrRecov$). The throughput of the transition $Sync$ represents the mean number of read and write operations performed by the system working in the *Degraded* state. For brevity's sake we omit to describe $GSPN_{OK}$.

To evaluate the queuing effects on read and write operations three simple QNs can be used.

QN_{CONTR} is the QN that describes the behavior of the disk array controller; QN_{OK} and QN_{DEGR} are the QNs that describe the behavior of the controller located on the physical disk array machine in the *Ok* and *Degraded* state respectively.

Of course the *Dead* state does not need a queue to be described since no activity is performed on the system when it is unavailable.

Experimental data on read and write operation are used to tune the queues. The request arrivals are supposed to be distributed as a Poissonian process so that the three queues

may be approximated by M/M/1 queues.

To obtain the RWMRT of the system, the mean of the distribution evaluated by QN_{CONTR} is corrected by using the probabilities of having a degraded or a full working system from the results of the analysis of FT_{DEGR} and FT_{DEAD}. The resulting distributions model the request arrivals to QN_{DEGR} and QN_{OK} whose service times are assigned by using the inverse of the throughput of the $Sync$ transitions of $GSPN_{OK}$ and $GSPN_{DEGR}$ respectively.

Thus a complete performability model of the system consists of a composition of different submodels. The semantics of this composition requires exchanges of information (results) between the submodels in order to obtain the automated solution of the RAID model.

A more complex case study could require to model different QN policies that are heavy to model by GSPN. In this case the features of the OsMoSys/DrawNET framework would be better exploited but a simple case study helps to illustrate the steps needed when using the framework.

IV. The OsMoSys/DrawNET Approach to Multisolution

The RAID model discussed in Section III needs adequate means to be effectively solved in an automated way, since many steps have to be performed in the right order and a flow of data must be controlled to obtain the required results. In this Section we introduce the OsMoSys/DrawNET approach to multisolution that is based on the concept of **orchestration**.

Orchestration is the arrangement and/or control of heterogeneous and autonomous entities that must cooperate to achieve a common goal. In computer science orchestration describes how applications and services can interact, including the business logic and execution order of the interactions. This allows to facilitate the construction of composite applications and services from a number of distributed and autonomous software components. A meaningful case of orchestration from the IT field is given by Web Services composition [15] which requires that proper workflow languages and execution engines are defined and developed to describe and execute a process flow between services, controlled by a single party.

The OsMoSys/DrawNET approach copes with multisolution by orchestration of modeling techniques and analysis/simulation tools. This is accomplished in two stages respectively, described in the next two subsections.

A. Bridge Formalisms and Bridge Operators

The OsMoSys/DrawNET framework is based on metamodeling and object orientation concepts that allow compositionality and extendibility in the modeling process. The underlying modeling methodology is introduced in [21]. It is centered on the definitions of *Meta-Formalisms*, *Model Meta-Classes* and *Model Classes*.

A **Meta-Formalism** is a language used to describe formalisms, in particular the Formalism Definition Language (FDL) is the Meta-Formalism that defines the primitives used to introduce different modeling formalisms (PNs, FTs, QNs, etc.) in the framework. In more details FDL defines:

- *E*: the set of *element types* by which the formalism is composed. For example in the case of PNs $E = \{place, transition, arc\}$.
- P_{FDL}: the set of properties that can be associated to the various elements. For example *token, weights and priorities*.
- A set of relations that couple elements, properties and their types sets (i.e $Real, String, Integer$ etc.).

A **Model Meta-Class** is a modeling formalism expressed by means of the above mentioned primitives.

A **Model Class** must be compliant with a Model Meta-Class and describes a family of models having the same structure expressed by means of the primitives of its Model Meta-Class. A Model Class may have an interface consisting of a subsets of its elements. A Model Class is not a concrete entity and it must be instantiated to obtain a concrete model, called Model Object.

Let us introduce **Bridge Formalisms**. A Bridge Formalism is defined by means of the FDL, too: it is a Model Meta-Class whose aim is to allow the development of models composed by heterogeneous submodels. Fig. 3 shows the final model of the RAID example presented in Section III.

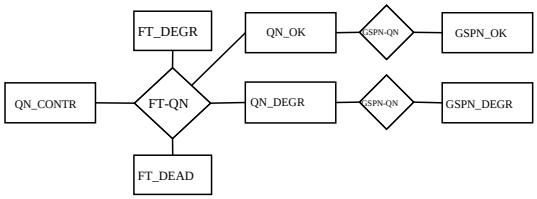

Fig. 3. The Raid Model

This model is built by defining a Bridge Formalism which allows to compose FTs, GSPNs and QNs submodels. In the figure the squares encapsulate the Model Classes, i.e. the submodels. The RAID model consists of seven submodels according to the description given in Section III; they describe a generic behaviour and must be instantiated by assigning values to the submodels parameters to obtain the model of a real system to analyze. FT_{DEAD}, FT_{DEGR}, $GSPN_{OK}$, $GSPN_{DEGR}$, QN_{OK}, QN_{DEGR}, and QN_{CONTR} are Model Classes compliant with their own Meta-Classes (FT, GSPN and QN).

The rhombuses in Fig. 3 represent Bridge Operators. They implement the semantics of the connections between submodels. The arcs linking a Bridge Operator to a Model Class are connected to proper elements of the Model Class that belong to its interface. In Fig. 3 two types of operator are used: a Bridge Operator between FT and QN submodels ($FT-QN$) and a Bridge Operator between QN and GSPN submodels ($GSPN-QN$).

The example shows that a composed multi-formalism model has a graph based structure whose nodes are submodels. The operators represent the semantics of the connections between

submodels. The set *E* of a *Bridge Formalism* contains Bridge Operators, arcs, submodels and references to the interface elements of the Model Classes whose objects must be connected. Consequently, a Bridge Formalism must be defined according to: a) the combination of Model Meta-Classes the user want to use in building a multi-formalism model; b) the set of interface elements; c) the set of the available operators allowing the interaction between models expressed by means of the specified Model Meta-Classes.

Fig. 4 illustrates the connections between the $FT - QN$ operator and the queues modeling the behaviour of the controllers. Notice that this composition is not a queuing network. The connections with the FTs Model Classes are shadowed and represented by the dashed square and the related arc.

The Bridge Operator is an element type belonging to the set *E* of the Bridge Formalism. The FDL allows to associate to each operator and to each arc a set of properties (belonging to P_{FDL}) which are the information needed to implement the semantics of the interaction between the submodels. In Fig. 4 the operator $FT-QN$ acts as a function which has three input parameters, one of them from QN_{CONTR} (this information is a property of the arc from the queue to the operator) and two output parameters, one to QN_{OK} and one to QN_{DEGR} (defined by the properties of the two output arcs). The output parameters are calculated by the operator using the input from QN_{CONTR} and the probabilities a_1 and a_2 resulting from the analysis of FT_{DEAD} and FT_{DEGR} respectively.

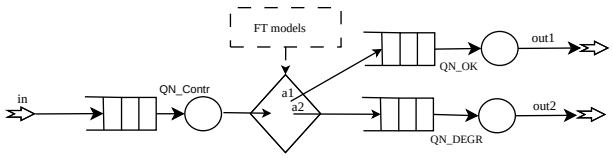

Fig. 4. The QN-FT Bridge Operator of the RAID model

In the development stage of a system models, the Bridge Formalisms are languages which define the interactions between the submodels, the routing of the information that must be implemented and the operations that must be performed in order to *orchestrate* different modeling formalism and techniques. The graphical representation of a composed model obtained by means of a Bridge Formalisms is compiled by the DrawNET tool that generates a textual XML based definition of the model, including the information provided by the Bridge Operators and arcs. This textual definition of the model is used in the solution stage.

B. Orchestration of solvers

The automated solution and analysis of a multiformalism model is an hard task since it requires that a flow of data and activities is controlled according to a well defined sequence of steps. Let us consider the interactions between the GSPN and QN Model Class in the RAID example on the right of Fig. 3. The semantics of the connections require to:

1) solve the $GSPN_{DEGR}$ ($GSPN_{OK}$) model;
2) retrieve the throughput of the $Sync$ transition;
3) calculate the inverse of this throughput;
4) instantiate the QN_{DEGR} (QN_{OK}) model by assigning to the service time the resulting value.

The finite, ordered sequence of steps needed to analyze the entire model can be considered an algorithm which is expressed by means of a proper workflow language and whose execution is a process named **solution process**. The processor able to execute a solution process is a workflow engine. In Fig. 5 the resulting architecture of the OsMoSys/DraNET framework is sketched.

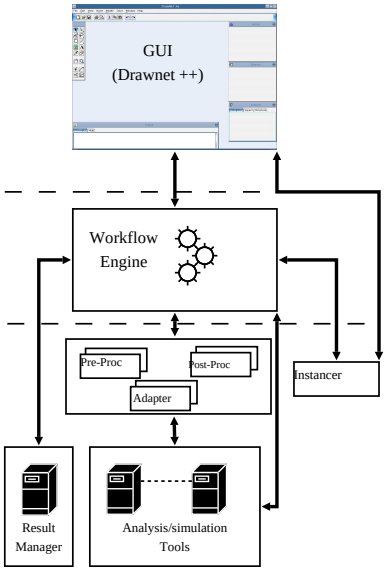

Fig. 5. Architecture of the OsMoSys/DrawNET framework

The Analysis/Simulation Tools are the solvers involved in the solution processes. They can be processed on different machines and their execution is requested by the workflow engine. The Adapters integrate the solvers into the framework, translating input and output formats to an intermediate (XML) notation in order to allow the data exchange among different solvers.

Pre-processors (Pre-Proc in Fig.5) are software tools used, if it is necessary, to translate the OsMoSys model descriptions to different formats. Post-Processors (post-Proc in Fig.5) are similar to Pre-processors. The only difference is that they are used *during* the solution process if some transformation is needed in order to compose and elaborate submodels to build a new intermediate model. The Result Manager is a software application in charge of processing intermediate results to calculate the results requested by the user. It also allows, with the aid of the *Instancer*, to instantiate a (sub)model *during* the solution process if it is necessary (e.g. the QN submodels have to be instantiated after that the values of the service rates are available). The Result Manager will be briefly introduced later, in Section V-B.

The **Solution Process Definition Language (SPDL)** is an XML-based workflow language developed to orchestrate the solvers. It enables task-sharing for a distributed computing

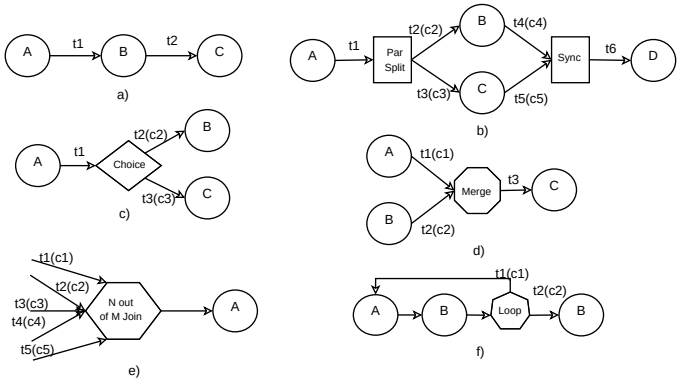

Fig. 6. Solution Patterns

by a combination of analysis/simulation tools. Using SPDL, a solution process is formally described that will take place across a set of solvers in such a way that any cooperating entity can perform one or more steps in the process the same way.

SPDL allows to define an executable solution process by enabling: a) the activation of a tool or solver, b) data routing among applications during the solution process, and c) the definition of complex execution patterns.

The first point is needed to perform the instantiation and the execution of an application during the solution process. The language allows to define the parameters needed to invoke applications and the names and the paths of input/output files.

The second point is needed to provide the right input data to each application before its execution.

The last point is needed since applications have to be orchestrated according to execution path involving the definition of non-trivial data path. At the state, SPDL supports the patterns reported in Fig. 6. They refers to the patterns defined in [19], [20].

In the figure, circles represent *activities* to be executed, (mostly activations of applications); edges, called *transitions*, describe how activities are linked, defining the control flow and the data flow path for the process. A predicate expression named *condition* should be associated to a transition. It is evaluated to state if a path can be activated or not. The patterns supported by SPDL are (the letter refers to Fig. 6):

a) *Sequence*: Applications are activated in sequence. Output data of an application should be the input data of the next application that must be executed.
b) *Parallel Split*: Conditions c_2 and c_3 evaluate *true* and the applications A and B are executed in parallel.
b) *Synchronization*: The $Sync$ operator activates the transition t_6 only after that all the incoming transitions have been activated (not necessarily at the same moment). This pattern is used to describe the synchronization (*join*) of two or more execution paths.
c) *Exclusive Choice*: The operator $Choice$ activates transition t_2 or transition t_3.
c) *Multiple Choice*: The $Choice$ operator can activate one or more outgoing transitions, depending on the value of the associated conditions.
d) *Multiple Merge*: The pattern merges many execution paths without synchronizing. The transition t_1 and t_2 are not simultaneously activated. C is executed two times, one for each activation of the incoming transition of the $Merge$ operator
d) *Discriminator*: The pattern merges many execution paths without synchronization but C is executed only once.
e) $N - out - of - M$ *Join*: The $N - out - of - M$ Join operator performs partial synchronization over incoming transitions. The activity A is executed only once after that the synchronization has occurred.
f) *Arbitrary Cycles*: This pattern is used to describe loops. The sequence of activities A-B is repeated depending on the value of the conditions c_1 and c_2.

A SPDL solution process starts by a declarative section. The basic elements involved in a solution process that must be declared by its SPDL definition are:
Participant, *Application*, *Variable*, *Activity*, *Condition* and *Transition*.

A Participant is a physical node where Applications are executed. Its definition provides the information needed to schedule and invoke the execution of an application.

An Application represents a tool, solver or framework component that can be invoked and executed on a Participant involved in the solution process. Usually applications needs parameters and input files in order to be executed. They must also be defined as *Formal parameters* for applications in the SPDL declarative section. When an application is invoked inside the solution process, each formal parameter must be replaced by the related actual parameter.

Variables are used to store process data when needed. Usual programming language types can be associated to variables: integer, real, boolean, double and string.

Activities represent the basic units of work for a solution process and they are represented by the nodes of its data and control flow graph. They must be *activated* in order to accomplish their task.

Conditions are predicate evaluated over variables.

Transitions define how activities are linked together and they are represented by the edges of the data and control flow graph of the solution process. Each transition is linked to two activities, called *from* and *to* activity respectively.

The *from* activity is an outgoing edge, the *to* activity is an incoming edge. Transitions may be associated to a condition. A Transition is activated when its *from* Activity ends its execution and if the condition predicate defined on it evaluates true. After the activation of a transition, its *to* activity becomes a candidate for a subsequent activation. Similar to workflow languages, to allow the definition of the patterns previously described, different kind of split and join operators are defined on activities [1] (e.g., AND splits and AND joins implementing parallel activation and synchronization of

activities respectively).

Few lines from the SPDL solution process of the RAID example are listed in Fig. 7. This code belongs to the declarative section of the SPDL program and refers to the analysis of the $GSPN_{DEGR}$ model whose aim is to get the throughput of the transition $Sync$. The first part of the listing reported by the figure contains the declaration of the Participant on which a solver able to analyze/simulate GSPN model can be invoked. Notice that the location of the node must be specified. The second part of the listing contains the declaration of the Application that must be executed on the Participant. In this case it is the $newSO$ tool of the $GreatSPN$ package. The definition of the application also specifies the path to be used to invoke the application. The application needs to know the model to analyze (the IN formal parameter) and the name of a file (the INOUT formal parameter). The third part of the listing is the declaration of an activity that requests the execution of $newSO$. The definition of the activity specifies the AND Join condition and the AND Split condition. Moreover, the actual parameters "$Great/Raid5/GSPN_{DEGR}$" and "$Great/Raid5/GSPN_{DEGR}.net$" are provided. They are the path of the OsMoSys/DrawNET textual description of the $GSPN_{DEGR}$ model and of the GreatSPN file of the GSPN model respectively.

```
<SolutionProcess  name="Raid">
 <ProcessHeader  name="Raid"/>
  <Participants  nameset ="AllParticipants ">
    <Participant name="local" location="127.0.0.1"
     applicationsref  ="applset _on_ localhost " />
  </Participants>
...
  <Application name="  newso " path="./Great/  newSO "
    performer="local">
   < FormalParameters  >
    <  FormalParameter  name="  modelname  " datatype ="var" type="IN" />
    <  FormalParameter  name="  Gfile " datatype ="file" type="  INOUT " />
   </ FormalParameters  >
  </Application>


  <Activity name="  GspnDEGR  "
    performer="local" type="normal">
   < JoinType  type="AND" />
   < SplitType  type="AND" />
   < ActivityApplication  name="  newso ">
    <  ActualParameters  >
     <  ActualParameter  name="  modelname  " datatype ="var" type="IN"
       value="Great/  Raid5 /GSPN _DEGR  "/>
     <  ActualParameter  name="  Gfile " datatype ="file" type="  INOUT "
       value="Great/  Raid5 /GSPN _DEGR.net  " />
    </  ActualParameters  >
   </ ActivityApplication  >
  </Activity>
...
</SolutionProcess  name="Raid">
```

Fig. 7. SPDL example from the code of the RAID solution process

V. Performance Indices and Results Handling

The orchestration of solvers makes sense only if performance indices are defined on the model that must be evaluated. The compositional nature of the model and multiformalism put the user requests for indices evaluation on a (logical) level that can be far from the level of the analysis and simulation tools used during the solution process. Major concerns handling performance indices in multiformalism multisolution environments are:

- how to specify the indices to be evaluated on the model and how to translate them from the user level to the solvers level;
- how to retrieve and combine the results obtained by the execution of more analysis/simulation tools to produce the answer to the user's request;
- how to present the final results to the user.

In the case of the RAID example, the final user want to know the mean response time of read and write operations (RWMRT). This means that he/she wants to evaluate a measure on the global model that requires to evaluate and combine some partial results, as we have explained in the previous Sections. The user could be able to write the SPDL solution process of the model, but he/she could also be interested in using a "black box" approach and developing the model by using predefined Class Models, Bridge Formalisms and Operators. He/she could be not aware of the different formalisms and modeling techniques used to develop the submodels. In this case the development of the SPDL program of the solution process can be automated by means of predefined SPDL skeletons and information gathered from the submodels, the Bridge Formalism and the Bridge Operators, but it is necessary that information about the performance indices are available too, and that proper mechanisms are defined to answer the above mentioned questions. The OsMoSys/DrawNET solution to this problem is based on the concept of **query** and the development of a set of languages oriented to indices and results handling, as introduced in the following.

A. Queries and Results Definition

In order to solve a model, a query document is associated by the DrawNET tool to the model textual description. A *query* specifies the performance indices to be evaluated on the model. The performance indices mentioned in the query must belong to a predefined set of indices that are admissible for the Model Meta-Class used to develop the model, i.e. the set depends on the modeling language to which it is associated and on the availability of proper solvers in the framework. The set of admissible indices of a Model Meta-Class is defined by means of the Results Definition Language (RDL). A RDL document related to a given Model Meta-Class contains the correspondences between the elements of the Meta-Class and the indices that may be calculated.

In Fig. 8 two examples of RDL documents are shown. We remember that the Model Meta-Class of the RAID model is a Bridge Formalism ($Bridge - FT - QN - GSPN$ in the following) and that the Model Meta-Class of $GSPN_{DEGR}$ and $GSPN_{OK}$ is the GSPN formalism expressed by means of the FDL primitives. The examples in Fig. 8 are related to the GSPN and the $Bridge-FT-QN-GSPN$ Meta-Classes respectively.

In this example, the RDL associated to the GSPN formalism specifies some indices belonging to two classes: global indices (i.e. they refer to the GSPN model) and indices that are

```
<rdl main=" GSPN ">
 < elementType   name=" GSPN ">
  < resultType   name="bounded"/>
  < resultType   name=" islive "/>
 < elementType   name="Transition">
  < resultType   name="throughput"/>
  < resultType   name=" covbytinv  "/>
 </ elementType  >
 < elementType   name="Place">
  < resultType   name="mean"/>
  < resultType   name=" covbypinv  "/>
 </ elementType  >
  < resultType   name=" Tinv "/>
  < resultType   name=" Pinv "/>
 </ elementType  >
</ rdl>
```

```
<rdl main=" BridgeFT -QN -GSPN ">
 < elementType   name=" BridgeFT -QN -GSPN ">
  < resultType   name=" meanTr  "/>
  < aggregateType   name=" trOnPath  " >
    <  resultType   name=" meanValue  "/>
    <  propertyType   name=" inputPlace  "
            default="" type="        elementRef  "/>
    <  propertyType   name=" outputPlace  "
            default="" type="        elementRef  "/>
  </  aggregateType  >
 </ elementType  >
</ rdl>
```

Fig. 8. RDL examples

```
<mql rdl="BridgeFT -GSPN -QN.rdl "
   main="RAID_MODEL">
<element name="RAID_MODEL"
      type="$   BridgeFT -GSPN -QN ">
 <aggregate name="    trOnPathOK  "
       type="    trOnPath  ">
  <result name="    meanValue  "/>
  <property name="    inputNode  "
          value="   QN_CONTR  "/>
  <property name="    outputNode  "
    value="  bridgeGSPN  _QN1.QN_OK  "/>
 </aggregate>
 <aggregate name="    trOnPathDegradated    "
       type="    trOnPath  ">
  <result name="    meanValue  "/>
  <property name="    inputNode  "
          value="   QN_CONTR  "/>
  <property name="    outputNode  "
    value="  bridgeGSPN  _QN2.QN_DEGR/>
 </aggregate>
</element>
</mql>
```

```
<rsl rdl="bridgeFT -GSPN -QN.rdl ">
<frame base="false" label="atomic">
 <element name="RAID_MODEL"
       type="   BridgeFT -GSPN -QN ">
  <aggregate name="    trOnPathOK  "
        type="    trOnPath  ">
   <result format="single"
        name="    meanValue  ">
    <value  val="5.44e          -04"/>
   </result>
  </aggregate>
  <aggregate name="    trOnPathDegraded      "
        type="    trOnPath  ">
   <result format="single"
        name="    meanValue  ">
    <value  val="6.64e          -04"/>
   </result>
  </aggregate>
 </element>
</frame>
</rsl>
```

Fig. 9. Query and results document

```
<mql rdldef ="GSPN.rdl  ">
 <element name="   GSPN _DEGR      " type="$ GSPN ">
 <element name="Sync" type="Transition">
   <result name="Throughput">
 </element>
</element>
```

Fig. 10. L1 Query for the Sync throughput

associated to elements of the formalism (i.e. place, transition, arc). The two global indices named $bounded$ and $islive$ are used if the boundness and liveness properties of a net must be checked. The indices associated to transition elements are named $throughput$ and $covbytinv$, they are used if the throughput and the presence of a transition in at least one T-invariant must be evaluated. The indices associated to place elements are named $mean$ and $covbypinv$, they are used if the mean of the token distribution and the presence of a place in at least one P-invariant must be evaluated.

The RDL associated to the $Bridge-FT-QN-GSPN$ Bridge Formalism uses an Aggregate Element type to specify the index $trMean$. An *Aggregate Element* is used to specify properties related to *subset* of elements of a Model Meta-Class. $trMean$ in Fig. 8 is associated to the Aggregate Element $trOnPath$ and must be obtained evaluating intermediate results ($meanValue$) over the path containing the elements named $inputPlace$ and $outputPlace$ (the operation to perform on them is defined by the Bridge Operator).

A query associated to a (sub)model specifies what indices must be evaluated on the (sub)model among the ones reported in the RDL document of the (sub)model Meta-Class. The query needed to request the value of the throughput of the *Sync* transitions of the $GSPN_{DEGR}$ and $GSPN_{OK}$ submodels is shown in Fig.10. The query associated to the RAID model is on the left of Fig.9. The queries are expressed by means of the Model Query Language (MQL).

The simple query in Fig.10 is used to request the evaluation of an intermediate result. The first line says that the query is based on the GSPN.rdl document, the second line says that the query is associated to the GSPN model named $GSPN_{DEGR}$ (a model is an element according to the FDL), the remaining lines say that the query asks for the evaluation of the throughput of the transition $Sync$.

The query on the left of Fig.9 is used to request the evaluation of the RWMRT both in the *Degraded* and *Ok* cases. The requested measure corresponds to the $meanTr$ index associated to the Aggregate Element defined in the $Bridge-FT-QN-GSPN$.rdl document. The query specifies the $inputPlace$ and $outputPlace$ elements in the two cases.

Having introduced the RDL and MQL languages, we give an original definition of solver and multisolution.
A *solver* is a function $Sol : MQL \rightarrow RSL$ that associates a solution $rsl \in RSL$ to a query $mql \in MQL$.
The solution process of a model is a **singlesolution** process if a single solver Sol exists that is able to compute $rsl = Sol(mdl)$ in a single pass, where *mdl* is the textual definition of the model generated by DrawNET and expressed by means of the Modeling Definition Language (MDL). The solution process of a model is a **multisolution** process if $rsl = Sol(mdl)$ is computed by means of K singlesolution steps.

In general, the following four cases can occur:

- **Flat Model - Explicit Query (FMEQ)**: The model does not contain submodels (it is a *flat* model), it is expressed by means of a single formalism (that is not a bridge formalism) and its related query Q requires only the direct evaluation of performance indices explicitly specified by the query. The query Q is said to be an **explicit query**.
- **Composed Model - Explicit Query (CMEQ)**: A Bridge Formalism is used to describe the model that consists of more submodels and the query Q is an explicit query.
- **Flat Model - Implicit Query (FMIQ)**: The model is a flat model but the query Q must be translated into *n* different queries according to the following translation function: $Q_T(mql) = \{mql_i\}(i = 1, .., n)$. The query Q is said to be an **implicit query**. This case occurs if the user is not aware of the solution process required to solve the model and/or the intermediate results that

are necessary to evaluate the specified indices do not explicitly refer to elements of the Model Meta-Class used to develop the model (e.g. the case of Aggregate Elements).

- **Composed Model - Implicit Query (CMIQ)**: A Bridge Formalism is used to describe the model that consists of more submodels and the query Q is an implicit query. The query on the left of Fig.9 is a CMIQ.

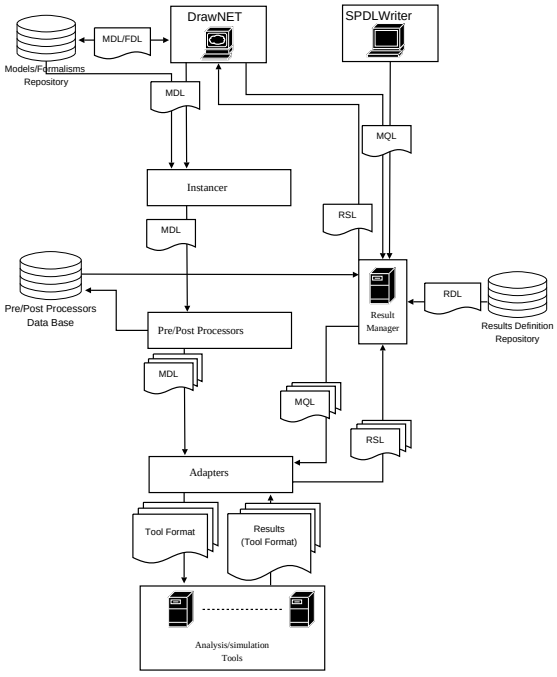

Fig. 11. OMF data flow

In order to clarify the interactions among the models the languages and the associated tools mentioned above, in Fig. 11 the general data flow in the OsMoSys/DrawNET framework is depicted. DrawNET accesses a models/formalisms data base to retrieve data (in MDL and FDL languages) needed to build a model. The model, expressed in MDL is submitted to the instancer in order to build model objects. The instancer receives the model and retrieves the definition of the model classes needed to instantiate the model. The model is then sent to Pre-Post processors and it is manipulated according to the solution process, before being dispatched (in submodels, each solvable by one solver) to the Adapters. The Adapters convert the submodels into the native solver formats and pass them to the solvers. On the other side, queries (in MQL) generated by DrawNET or included in the solution process written by the SPDLWriter (a Graphical User Interface used to define solution processes) are analyzed and transformed by the Result Manager, which in turn queries adapters and retrieves results in the RSL format. In order to validate queries and results format, a repository (Result Definition Repository) is used which contains the definitions of possible results available for a certain formalism (in RDL). Finally the Result Manager sends the requested results to DrawNET for the presentation.

B. Retrieving Results

The introduction of the RDL and MQL languages proposes a solution to the problem of specifying the indices to be evaluated on the model and translating them from the user level to the solvers level. Here we briefly describe how the query are processed and results are collected.

The explicit queries can be processed by enacting the solution process without generating intermediate queries. An implicit query requires that n intermediate queries are generated to enact the solution process.

The Result Manager (Fig. 5) is the software module in charge of automate the processing of the queries and collect the results produced by the execution of the solution process. From the point of view of the solution process and of the workflow engine, the Result Manager is an Application.

At the state of our research a first prototype implementation of the Result Manager is available. Its architecture consists of two layers (*L1* and *L2*).

L2 layer receives the query Q associated to the user's model and builds the final results according to Q, if that be the case on the basis of the intermediate results provided by the *L1*. *L2* also performs, if needed, the translation function Q_T. We remember that information about how to combine the intermediate results are provided by the Bridge Operator and that they are codified in the solution process.

L1 layer receives the queries to be processed from *L2* and collects the intermediate results needed to instantiate the submodels. At the end of the solution process it collects results from the solvers outputs passing them to the *L2* layer in the DrawNET format for results, in order to allow their presentation to the user.

Notice that in simple cases (e.g. the FMEQ) the *L1* layer is not involved in the solution process.

Now, we briefly show some experimental results obtained by solving the RAID model. The testbed architecture consists of an Hewlett-Packard SMART Array with two disk array local controllers. The array was equipped with three SCSI disks and a single local controller. A Linux node was equipped with the interface controller to the SMART Array box. Caching mechanisms on the local controller and on the interface controller were disabled. The stripe length was setted to 8 kbytes and all measures refers to 2GByte sequential read/write operations. Tab. I resumes some results related to the submodels and to the whole model. Times are expressed in milliseconds (msec). The first column contains the name of the index to be evaluated, the second column contains the value obtained by executing the solution process (if any); the third column contains the value measured on the real system and the last column contains the percentage variation between the values of the two previous columns.

The Read (Write) OP./Disk indices represent the time needed to write a whole stripe on each disk during Read (Write) operations. These values were measured off-line on a single disk disk in order to provide parameters to instantiate

TABLE I
SOME EXPERIMENTAL RESULTS

Perf. Index	*Model*	*Value measured on real system*	*Variation*
Read Op./disk	-	0.827	-
Write Op./disk	-	0.891	-
1/$Sync$ Degrad	0.520	0.503	3.4%
1/$Sync$ Ok	0.449	0.440	2.0%
$t_r onPathDegraded$	0.664	0.639	3.9%
$t_r onPathOK$	0.544	0.532	2.3%
% ok	99.997%	-	-
t_r Average	0.544	-	-

the RAID model. The 1/$Sync$ Degrad (Ok) parameter represents the inverse of the $Sync$ parameter in the $GSPN_{DEGR}$ ($GSPN_{OK}$) model. The parameter $t_r onPathDegraded(Ok)$ represents the mean response of 8 Kbytes read and write operations when the system is in the Degraded (Ok) state. $\%\ Ok$ is the probability of having the system in the Ok state (it is calculated from the solution of the FT models of the system). Finally t_r Average is the mean value of the previous two t_r values. On the right of Fig. 9 the document containing the answer to the query reported on the left is shown. It is generated by the $L2$ layer of the Result Manager in the XML format that is read by the DrawNET tool. The document mirrors the query and contains the values of the $meanValue$ measures required on the Aggregate Elements $trOnPathOK$ and $trOnPathDegradated$.

VI. CONCLUSIONS

In this paper we have presented a new approach and a software architecture to deal with multisolution when analyzing multiformalism complex models. It is based on the orchestration of heterogeneous solvers by means of proper languages and middleware. A workflow language, the SPDL, is used to write complex solution processes. Furthermore a language enabling the construction and the processing of queries on the user's models is used to retrieve performance indices to be evaluated on a model by multiple interacting analysis techniques and tools. The software architecture enabling the orchestration consists of two main components: a workflow engine and a results manager. The workflow engine executes the steps needed to perform the actions required to produce the results, the results manager is in charge of processing the user's query and collecting the intermediate results in order to compose the answer. The resulting architecture is complex, but it is very flexible since it enables interoperability among very different solution methods and solvers. Moreover, it provides an effective modeling and analysis framework both to a modeler who is familiar with many different kinds of models and a modeler who wants to use existing building blocks to build models without knowing the details of the modeling formalisms. The first can easily choose the models that best suit a particular system and the kind of indices that are needed at each stage of the design, the second can use predefined procedures to analyze the final model.

REFERENCES

[1] R. Allen. *Workflow: An Introduction*. Extracted from the Workflow Handbook 2001, Workflow Management Coalition, Available at: http://www.wfmc.org/standards/docs.htm

[2] F. Bause,P. Buchholz, P. Kemper. *A Toolbox for Functional and Quantitative Analysis of DEDS*. In: Pujanger R., Savino N.N., Serra B. (eds): Quantitative Evaluation of Computing and Communication Systems, LNCS 1469, Springer-Verlag, pp.356359, 1998.

[3] H. Bohnenkamp, H. Hermanns1, J.P. Katoen and R. Klaren. *The Modest Modeling Tool and Its Implementation*. In: Computer Performance: Modelling Techniques and Tools -Tools for Evaluation of Stochastic Models, LNCS 2794, Springer-Verlag, pp. 116-133, 2003.

[4] G. Ciardo and A.S. Miner. *SMART: Simulation and Markovian Analyzer for Reliability and Timing*. In: Tools Descriptions from the 9th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation and the 7th International Workshop on Petri Nets and Performance Models, Saint Malo, France, pages 41-43. June 1997.

[5] G. Ciardo, R.L. Jones, A.S. Miner, and R. Siminiceanu. *SMART: Stochastic Model Analyzer for Reliability and Timing*. In: Tools of Aachen 2001 International Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems, pp. 29-34, Aachen, Germany, Sept. 2001.

[6] D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J.M. Doyle, W.H. Sanders, P.G. Webster. *The Möbius Framework and its Implementation*. IEEE Transactions on Software Engineering, 28(10):956969, 2002.

[7] J. de Lara, H. Vangheluwe, M. Alfonseca. *Meta-Modelling and Graph Grammars for Multi-Paradigm Modelling in $AToM^3$*. In: Software and System Modeling (SoSyM) Journal (Springer), Special Section on Graph Transformations and Visual Modeling Techniques.

[8] H. Garavel, F. Lang, and R. Mateescu. *An overview of CADP 2001*. In: Proc. of Tools Day, August 24, Brno, Czech Republic, 2002.

[9] H. Garavel, F. Lang, and R. Mateescu. *An overview of CADP 2001*. EASST Newsletter, 4:13-24, 2002.

[10] G. Gibson, W. Courtright II, M. Holland and J. Zelenka *RAIDframe: Rapid prototyping for disk arrays*. Computer Science Technical Report CMU-CS-95-2000, Carnegie Mellon University, 1955

[11] M. Gribaudo, A. Valente. *Framework for Graph-based Formalisms*. In: Proceeding of the first International Conference on Software Engineering Applied to Networking and Parallel Distributed Computing 2000 (SNPD'00), Reims, France, pp. 233-236, 2000.

[12] F. Moscato, N. Mazzocca, V. Vittorini. *Workflow Principles Applied to Multi-Solution Analysis of Dependable Distributed Systems*. In: Proc. of the 12th Euromicro Conf. on Parallel, Distributed and Network-based Processing, February 11-13, 2004, A Coruna, Spain. pp. 134-141.

[13] P.J. Mosterman, H. Vangheluwe. *Special Issue on Computer Automated Multi-Paradigm Modeling* In: ACM Transaction on Modeling and Computer Simulation, Vol. 12, No.4, October 2002.

[14] D.A. Patterson,G.A. Gibson and R. Katz. *A Case for Redundant Arrays of Inexpensive Disks (RAID)*. In: Proc. of the ACM SIGMOD International Conference on Data Management, Chicago, IL., June 1-3,1988, pp.109-116

[15] C. Peltz. *Web Services Orchestration and Choreography*. IEEE Computer, Vol.36, n.10, 2003, pp. 46-52.

[16] R.A. Sahner and K.S. Trivedi and A. Puliafito. *Performance and Reliability Analysis of Computer Systems; An Example-based Approach Using the SHARPE Software Package*. Kluwer Academic Publisher, 1996.

[17] W.H. Sanders, T. Courtney, D. Deavours, D. Daly, S. Derisavi, and V. Lam. *Multiformalism and Multisolution-method Modeling Frameworks: The Möbius Approach*. In: Proc. of the Symposium on Performance Evaluation - Stories and Perspectives. Vienna, Austria, December 5-6, 2003, pp. 241-256.

[18] K.S. Trivedi. *SHARPE 2002: Symbolic Hierarchical Automated Reliability and Performance Evaluator*. DSN 2002, pp. 544

[19] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. Distributed and Parallel Databases, 14(3), pages 5-51, July 2003

[20] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Advanced Workflow Patterns. In Proc. of 7th International Conference on Cooperative Information Systems , vol 1901 of Lecture Notes in Computer Science, pages 18-29. Springer-Verlag, Berlin, 2000.

[21] V. Vittorini, M. Iacono, N. Mazzocca, and G. Franceschinis. *OsMoSys: a new approach to multi-formalism modeling of systems*. Journal of Software and System Modeling, 3(1):68-81, March 2004.